

REMARKS

Favorable reconsideration of this application, as presently amended, is respectfully requested.

Claims 1-21 are pending in the present application. In the outstanding Office Action, the drawings were objected to as failing to comply with 37 CFR § 1.84(p)(5). The specification was objected to under 35 U.S.C. § 112, first paragraph. Claims 8-12 were rejected under 35 U.S.C. § 112, second paragraph. Claims 1-21 were rejected under 35 U.S.C. § 103(a) as unpatentable over Hetherington (U.S. Patent No. 6,275,810). The rejection of these claims is respectfully traversed.

Responsive to the objections to the drawings and specification, the drawings and specification have now been amended to clarify the terminology found objectionable in the Office Action. A substitute specification, replacement abstract, and marked-up versions thereof are submitted herewith. Replacement versions of Figures 3, 4, 5, and 6 are additionally submitted herewith. Figure 3 is amended to replace "locale-dependent" with --locale-specific-- (elements 304, 306, 308), and to replace "encoding" with --subprocess-- (elements 312, 314, 316). Figure 4 is amended to replace "locale-dependent" with --locale-specific-- (steps 408, 414), and to replace "encoding" with --subprocess-- (step 412). Figure 5 is amended to replace "user-dependent" with --user-specific-- (elements 504, 506, 516), and to replace "encoding" with --subprocess-- (elements 508, 514). Figure 6 is amended to replace "encoding" with --subprocess-- (step 608). The specification has been amended to reference element number 352 instead of the previous reference to element number 353. Additionally, the specification has been amended to reference element 220. Applicants respectfully submit that element 420 was

specifically described in the originally filed specification on page 7, line 22, and that element 614 was specifically described in the originally filed specification on page 9, line 3.

Attention is first directed to the rejection of independent Claim 8 under 35 U.S.C. § 112, first paragraph. As discussed above, the specification has been amended to clarify the terminology found objectionable in the Office Action. Additionally, Claims 8-11 have been amended to clarify the recitation of a computer-readable medium, and are thus allowable under 35 U.S.C. § 112. No new matter is added by this amendment.

Attention is now directed to the rejection of Claim 8 as unpatentable over Hetherington. Claim 8 recites a computer-readable medium comprising computer instructions that facilitate concurrent handling of subprocesses in a system that utilizes a global process, wherein the instructions, when executed, cause the system to perform the step of mapping a plurality of concurrent user-specific processes, wherein each user-specific process is mapped to virtual addresses that are equivalent to virtual addresses of the global process.

According to the Examiner, Hetherington teaches "instructions (set of instructions, col. 15 lines 60-65), mapping of a plurality of concurrent user-specific processes (mapped, col. 6 lines 58-67), global process (daemon)." According to the Examiner, Hetherington "does not explicitly teach user-specific process is mapped to virtual addresses that are equivalent to virtual addresses of the global process." Additionally, according to the Examiner, "It would have been obvious for one skilled in the art to recognize that virtual addresses are needed when a process spawning a child process (the endpoint executables are spawned by daemon 24a, col. 5, lines 2-10)." However, Applicants respectfully submit that there is no mention or suggestion of "mapping a plurality of concurrent user-specific processes, wherein each user-specific process is mapped to virtual addresses that are equivalent to virtual addresses of the global process" in

Hetherington. Rather, Hetherington mentions sets of instructions stored in random access memory, a daemon 24a which is responsible for endpoint login and for spawning application endpoint executables as part of a management agent on the endpoint, which is a machine running a system management framework client, and an Add Holiday Dialog panel which enables an administrator to define a holiday, specify the locales it is observed in, and to specify the calendar to which it is mapped. Assuming that the daemon is the recited "global process" (*See Office Action*, page 5, line 5), Hetherington neither teaches nor suggests "each user-specific process is mapped to virtual addresses that are equivalent to virtual addresses of the global process," nor does the Examiner explain how this feature is rendered obvious over Hetherington.


Therefore, Applicants respectfully request that this rejection of Claim 8 be withdrawn. With respect to Claims 9-11, these claims depend from Claim 8, and thus, Applicants respectfully request that the rejection of Claims 9-11 be withdrawn, at least for the same reason.

For reasons similar to those stated above with respect to Claim 8, Applicants respectfully request that the rejection of independent Claims 1, 12, and 15 also be withdrawn. With respect to Claims 2-7, 12-14, and 16-21, these claims depend from Claims 1, 12, and 15, respectively, and thus, Applicants respectfully request that the rejection of Claims 2-7, 12-14, and 16-21 be withdrawn, at least for the same reason.

In view of the foregoing comments, Applicants respectfully submit that the present amendment places the above-referenced application in condition for allowance, and thus, a swift allowance is respectfully requested so that the application may swiftly pass to issue.

Respectfully submitted,

Dated: September 4, 2003



Margo Livesay, Ph.D.
Registration No. 41,946

Customer No.: 26263
Sonnenschein Nath & Rosenthal
P.O. Box 061080
Wacker Drive Station
Chicago, Illinois 60606-1080
Telephone: 202/408-6348
Facsimile: 312/876-7457

09/12/3
9

A METHOD FOR ENABLING MULTIPLE CONCURRENT SUBPROCESS HANDLING ON A SYSTEM USING A GLOBAL PROCESS

BACKGROUND OF THE INVENTION

5

Field of the Invention

The present invention relates generally to computer systems, and more specifically to multiple concurrent subprocess handling on systems employing global processes using a communications protocol that does not require data serialization or deserialization.

Related Art

10 To understand the present invention, one must first understand the relationship between a

global ~~processes~~ process and its subprocesses. Consider the case where a global process is accessed by a variety of client or user processes. When the global process needs to analyze user-specific data for a variety of such users, it does so by way of a subprocess. In a concurrent subprocess system, many subprocesses of a single global process are simultaneously active.

15 Locales in a Unix environment provide a practical example of global processes and subprocesses. A locale defines a set of language conventions ~~by~~ relative to language territory and code set combinations. These conventions include information on character classification, case conversion, date and time representation, monetary symbols, and numeric representations. One can think of locales, as including all the information necessary to communicate relay
20 information in a given language. Thus, one sometimes speaks of locales in terms of region or language, such as a Dutch locale or a Japanese locale.

ANSI-C is a version of the C programming language that adheres to standards set forth by the American National Standards Institute. The Portable Operating System Interface for Unix (POSIX) is a standard that defines operating system devices. A common conglomeration of

these standards is known as the ANSI-C/POSIX standard. This standard utilizes a global locale environment. What this means is that each process or task is bounded to a single global locale.

A system having a global locale environment is illustrated in Figure 1, which illustrates a server system 100. The system 100 includes a central processing unit 102; a primary memory 104, such as a random access memory, for program execution; and a secondary memory 106, such as a disk, for storing programs and data that are not immediately needed for execution. Contained within memories 104, 106 are an operating system 108 that utilizes a global locale. Also pictured is the kernel 110, which is the core of the operating system 108.

The kernel 110 has a single process for each differing locale, 150, 152, 154, and a single process for each differing user or client, even if that client utilizes the same locale 156, 158, 160. These single processes 150, 152, 154, 156, 158, 160 are created regardless of the amount of processing functionality they share. That is, the processes' instructions could be 90% locale-independent and largely functionally similar to each of the other processes. Nevertheless, these separate processes must be created under a global locale standard, such as ANSI-C/POSIX.

Unfortunately, each such process consumes precious computing resources: memory, which is directly consumed, and processing power, which is indirectly consumed when many processes occupy substantial amounts of memory.

In contexts such as this and others, past techniques have attempted to serve concurrent subprocesses using a variety of communication techniques, such as remote procedure calls and remote method invocations. These methods are limited, however, because they require the serialization and de-serialization of all data, including pointers used between the processes and subprocesses. This is costly in terms of processing time. The present invention provides a better way.

INVENTION SUMMARY

Embodiments of the present invention provide methods and systems for enabling multi-subprocess handling on systems employing a global or master process. In a preferred embodiment, a virtual memory separator interfaces with a master process and an operating system kernel. This separator utilizes the master process in conjunction with user-specific processes to enable concurrent subprocess handling. The virtual memory separator maps each user-specific process so that it virtually overlays the masters processes' address space. In a preferred embodiment, this mapped user-specific process is partially empty because it contains only the data and coding to perform user-specific operations. When user-specific instructions are encountered in the master process, processing is transferred to the user-specific process. Data is efficiently transferred because each user-specific process has an interface that is identical to its master process: because the subprocesses virtually overlay the addresses of the master process, there is no need to serialize and de-serialize data, including pointer information. As such, the present invention provides for high performance communications in concurrent subprocessing environments.

BRIEF DESCRIPTION OF THE DRAWINGS ~~DRAWING DESCRIPTIONS~~

The accompanying drawings illustrate embodiments of the invention that will enable those skilled in the art to make and utilize the invention. In fact, these drawings, when considered with the written disclosure, will enable those skilled in the art to modify these embodiments in a manner that does not depart from the spirit and scope of the invention.

Figure 1 depicts a prior art computer system.

Figure 2 depicts a computer system in accordance with a preferred embodiment of the present invention.

Figure 3 is a conceptual illustration of interactions between a master process and its concurrent subprocesses in a preferred embodiment of the present invention.

Figure 4 is a flowchart illustrating steps used in an embodiment of the present invention.

Figure 5 is a conceptual illustration of interactions between a master process and its constituent concurrent subprocesses in an alternate preferred embodiment of the present invention.

Figure 6 depicts a computer system that provides for multiple subprocess handling on a system using a global process in accordance with an alternate preferred embodiment of the present invention.

DETAILED DESCRIPTION

Embodiments of the invention will now be described with reference to the accompanying drawings. It is understood that the invention is not limited to these embodiments, as the teachings contained herein will enable others to modify these embodiments without departing from the spirit and scope of the invention. Accordingly, the invention encompasses all such modifications that are described by the appended claims and their equivalents.

Overview of The Detailed Description

The preferred embodiments of the present invention enable a global process system to operate as concurrent, multiple subprocesses. In part, the detailed description that follows illustrates preferred embodiments that employ the present invention in a global locale environment. Those skilled in the art will appreciate, however, that the invention is not limited to locale-based embodiments. On the contrary, this disclosure will enable those skilled in the art to create versions of the invention that relate to a variety of master process/subprocess technologies.

The structure of the disclosure is as follows: A computer system embodying preferred embodiments of the present invention is discussed. See Figure 2. An illustration of a concurrent locale system as practiced in a preferred embodiment is next described. See Figure 3. The disclosure continues with a detailed ~~explanations~~ explanation of processing steps of in a preferred embodiment. See Figures 4A, 4B. Next an alternate preferred embodiment, one using adaptive virtual memory separation, is described. See Figures 5, 6.

Figure 2: Computer System

A computer system that enables concurrent locales in a global locale computer system is illustrated in Figure 2. This embodiment illustrates a system 200 that includes a client computer 202 connected to a server computer 204 via a network 206.

The client system 202 includes a standard array of components: a central processing unit 208; a user interface 210, which typically consists of a keyboard, a mouse, and a monitor; a modem 212, a primary memory 214, such as a random access memory, for program execution; and a secondary memory 216, such as a disk, for storing programs and data that are not immediately needed for execution. Contained within memories 214, 216 are an operating system 218 that contains instructions and data for managing the computer's resources and a client program 220.

Like the client system 202, the server system 204 contains a CPU 222 and an operating system 224 stored in its primary and secondary memories 226, 228. In this embodiment, the operating system contains a kernel 230 that uses a global locale system, such as that defined by the ANSI-C/POSIX standard. Those skilled in the art will appreciate that the invention is not limited to any particular operating system. Indeed, while a valuable aspect of the preferred embodiment is that it can extend global locale systems to concurrent locale systems, the

preferred embodiment can run on any operating system that has memory mapping capabilities. Interacting with the kernel 230 is a virtual memory separator 232, which can contain an adaptive virtual memory separation component 234. In one preferred embodiment, the virtual memory separator 232 interfaces with an unmodified ANSI-C/POSIX compliant kernel. Since the kernel 230 has not been modified, the virtual memory separator 232 is highly portable. This adaptive virtual memory separation component is explained in detail with reference to Figures 5, 6.

Figure 3: Conceptual Illustration of Concurrent Locales

Two conceptual illustrations of concurrent locales as practiced in a preferred embodiment are illustrated in Figure 3. A process is said to be virtual memory separator-enabled when it accesses the virtual memory separator to complete its constituent subprocessing. The VMS-enabled process allocates memory for a master process 302, which interacts with the kernel 230 and performs the non-locale ~~dependent~~ specific operations. The VMS-enabled process also creates subprocesses, each of which the virtual memory separator maps in a largely empty virtual process space. As illustrated, the subprocesses are locale-specific processes for a variety of locales. The separator maps a locale-specific process for each desired locale: e.g., one for Russian 304, one for French 306, and one for Spanish 308. As shown, these ~~locale-dependent~~ locale-specific processes 304, 306, 308 are mapped to perfectly overlay the addresses of the master process 302. Thus, the virtual addresses of the ~~locale-dependent~~ locale-specific processes mirror the virtual addresses of the master process. As such, the respective addresses for the locale-specific subprocess 310 of the master process 302 correspond to those of the locale-specific ~~subprocess~~ subprocesses 312, 314, 316 of the ~~locale-dependent~~ locale-specific processes 304, 306, 308. When the master process 302 encounters the ~~locale-dependent~~ locale-specific

subprocess 310, program execution is transferred to the desired locale via a respective communication channel 318, 320, 322, 324.

For example, if a Russian and a French thread of a particular master process 302 are running, the data necessary to perform ~~locale-dependent~~ locale-specific operations is transferred
5 via the respective communication channels 318, 320, 322. Since the locale-specific processes 304, 306, 308 share an identical interface with the master process 302, and since they are mapped to virtual memory locations identical to those of the master process 302, the preferred embodiment accomplishes this communication without serializing and de-serializing data. Once
10 the ~~locale-dependent~~ locale-specific processes 304, 306 have the data, the subprocesses 312, 314 perform the ~~locale-dependent~~ locale-specific operations. After this is done, data is returned to the master process 302 via the respective communication channels 318, 320, 322.

A conceptual illustration of concurrent locales having multiple users in a preferred embodiment is shown in the lower portion of Figure 3. Five clients or users are illustrated: three
15 utilize French locales ~~353~~ 352, 358, 360, one utilizes a Russian locale 354, and one utilizes a Japanese locale 356. In this visualization of the preferred embodiment, the virtual memory separator swaps in the desired ~~locale-dependent~~ locale-specific subprocesses 364, 366, 368, 370, 372 as needed. This illustration highlights that the preferred embodiment can accommodate multiple users having the same locale, i.e., the illustrated French users 352, 358, 360. For
20 example, the French client represented by block 352 would often need different data processed than the French client represented by block 360. When one considers that this data could take a variety of ~~phones~~ forms, such as a string ~~represented~~ representing addressing or billing information, binary data, or a keystroke event, one appreciates why this is the case.

Figure 4: Virtual Memory Separator Flowchart

The processing steps used in a preferred embodiment of the present invention to enable concurrent locales in a global locale system ~~is~~ are illustrated in Figure 4. The process begins when a user, such as client program 220, outputs data, such as a string, binary data, or a keyboard or mouse event, to the master process on the server side (step 402). If this data is from a new user, the separator maps a new ~~locale-dependent~~ locale-specific process to the same address as the VMS-enabled master process (steps 404, 406, 408). Else, processing continues directly in the master process, which performs locale-independent operations (step 410). When ~~locale-dependent~~ locale-specific operations are encountered in the master process, those operations are executed in the subprocesses of the respective ~~locale-dependent~~ locale-specific processes (steps 412, 414). For example, if a thread belonging to a Spanish client encounters locale-specific subprocesses of the master process, program control is transferred to the Spanish ~~locale-dependent~~ locale-specific process. Data is shared via the previously described communication channel and there is no need to serialize the data due to the identical interfaces shared by the master and ~~locale-dependent~~ locale-specific processes and the identically mapped virtual addressing. Once the ~~locale-dependent~~ locale-specific subprocess has the necessary data, the ~~locale-dependent~~ locale-specific operations are performed in the respective ~~locale-dependent~~ locale-specific subprocess. When the ~~locale-dependent~~ locale-specific operations are completed, control is returned to the master process (~~step~~ steps 414, 416, 418). In this manner, processing continues until complete. Upon completion, the virtual memory separator returns data to the user (steps 418, 420).

Figure 5: Conceptual Illustration of Concurrent Subprocesses using Adaptive Methodology

A conceptual illustration of concurrent locales in an alternate embodiment of the present invention is shown in Figure 5. The illustration is similar to the embodiment previously

described in that it utilizes a master process 502 that interacts with ~~user-dependent~~ user-specific processes 504, 506 that are mapped over the master process by the virtual memory separator. As mentioned above, these ~~user-dependent~~ user-specific processes 504, 506 could be ~~locale-dependent~~ locale-specific or part of another master process/subprocess scheme, such as those prevalent in service daemons. The embodiment differs from that described above in that it creates the ~~user-dependent~~ user-specific process "on-the-fly"--that is, when the user-specific encoding is encountered. For example, when user-specific encoding 508 is encountered in the master process 502, the virtual memory separator creates a communication channel 510, 512 to transfer data between the master process 502 and ~~locale-dependent~~ locale-specific process 508. The user-specific operations are then completed in the subprocess 514. Once that processing is done, control is returned to the master process 502. Processing here continues until other ~~user-dependent~~ user-specific functionality 516 is encountered. When that happens, an additional subprocess is "spun-off": a communication channel is again opened and the subprocess is created. The channel/subprocess 518 can be created in the same mapped ~~user-dependent~~ user-specific process 504, or it 520 can be created in a separate ~~user-dependent~~ user-specific process 506. Either technique allows for the servicing of subprocesses within a master or global process environment. And because the subprocesses are mapped to virtual addresses that are identical to the master process and use the same interface as the master process, the embodiment avoids the costly serialization/de-serialization processing overhead that plagues conventional systems.

Figure 6: Adaptive Virtual Memory Separator Flowchart

The processing steps used in an alternative preferred embodiment of the present invention to enable concurrent subprocesses in a global process system ~~is~~ are illustrated in Figure 6, which further explains the adaptive concept explained with reference to Figure 5. Processing begins

when a user outputs data to a VMS-enabled master process, where user-independent operations are performed (steps 602, 604, 606). When user-specific encoding is encountered, the master process creates a user-specific process that the virtual memory separator maps to a user-specific process (~~step~~ steps 608, 610). As in the embodiment explained above, the user-specific process
5 shares the same interface and virtual addressing with its master process. In this embodiment, the user-specific process is largely an empty space--excepting the area that contains the user-specific subprocess, which is defined by user-specific instructions. Next, the user-specific instructions are executed in the user-specific subprocess (step 612). When this is done, processing is returned to the master process. Processing continues in this fashion until complete (steps 614,
10 606, 608, 610, 612), at which point data is returned to the client (steps 616, 618).

The Spirit and Scope of the Invention is Broad: It is not limited to the described embodiments

The above description will enable those skilled in the art to make numerous modifications to the described embodiments without departing from the ~~spirit~~ spirit and scope of the claimed invention. Indeed, the chosen embodiments were selected so others could best
15 utilize the invention by making such modifications to tailor the invention to their particular needs. By using the inventive concepts disclosed herein, those skilled in the art will be able to create further such embodiments that, although not explicitly disclosed, are implicitly disclosed by the concepts of the invention. The detailed description therefore should not be read as limiting the invention to the embodiments explained herein.

20 For example, this disclosure illustrated a computer system having a distinct server and client computer. Those skilled in the art will realize that the invention can be practiced on a single computer that uses master/subprocess handling.

Descriptive terms used in the disclosure also do not limit the invention. For example, the terms "master process," "subprocess," "global locale system," "locale-specific process," and "user-specific subprocesses" were all used to help describe preferred embodiments and to provide an enabling disclosure. While helpful for an understanding of the invention, these terms should not be read to limit the invention, as doing so would elevate form over substance: For these descriptive terms do not singularly define any embodiment of the invention, let alone the invention itself. Rather, it is the properties that these terms represent, in conjunction with the entire disclosure, that are important. So long as the inventive properties are being used, it makes little difference what terminology is invoked. For example, although one preferred embodiment explained the invention's use of a global locale system, the inventive concepts extend far beyond that area. As mentioned previously, the embodiments could be extended to a variety of master/subprocess concepts, such as service daemons.

Likewise, although portions of the disclosure referred to the ANSI-C/POSIX standard, those skilled in the art will recognize that the inventive concepts are not constrained to an operating system that adheres to this standard. On the contrary, embodiments can be extended to any operating system that employs memory mapping. Thus, the invention can be used in a variety of operating environments, such as UNIX, MacOS, LINUX, WINDOWS NT, WINDOWS 95/98/2000, the X-Windowing system or any JAVA runtime environment, which refers to the operating environment typified by a JAVA virtual machine and associated JAVA class libraries. JAVA is a registered trademark of SUN MICROSYSTEMS, INC.

The present invention is also not limited to any particular CPU or processing technology. Similarly, although the inventive concepts were described in part as being contained within random access memory and a hard disk, the present invention is not limited to these

devices. Those skilled in the art will recognize that these concepts can also be stored and invoked from any media that can store data or have data read from it. Examples of such media include floppy disks, magnetic tapes, phase discs, carrier waves sent across a network, and various forms of ROM, such as DVDs and CDs. The present invention anticipates the use of all
5 computer readable media.

Infringement is also not escaped by altering the order of the steps as they appear in the appended claims--for unless explicitly declared otherwise in the claim, the order set forth is of no significance. The invention therefore is not circumvented by alterations in terminology or other modifications that read on the appended claims and their equivalents.

ABSTRACT

A method and system for enabling multi-subprocess handling on computer systems that employ a global process. A virtual memory separator is provided as part of an operating system to interface with a master process and a kernel of the operating system. The separator maps user-specific processes to virtual address spaces that mirror that of the global process. These user-specific processes are empty spaces, excepting their interface -- which is identical to that of the global process -- and instructions necessary to carry out user-specific processing. When ~~user-dependent~~ user-specific operations are encountered in the global process, execution is transferred to a respective user-specific process. Since each user-specific process shares addresses and interfaces with the global process, data can be exchanged between them without serialization, which reduces processing overhead.